Лабораторная работа № 6

по дисциплине

“Объектно-ориентированное программирование”

Выполнил студент

группы БИБ1901

Бушеленков Роман

Москва 2021

**Цель работы:** реализовать возможность рисования фрактала с несколькими фоновыми потоками.

**Задачи:**

1. Создайте подкласс SwingWorker с именем FractalWorker, который будет внутренним классом FractalExplorer.

2. Класс FractalWorker

3. Метод doInBackground()

4. Метод done()

5. Функция void enableUI(boolean val)

6. Изменить реализацию mouse-listener

**Результат выполнения работы:**

package lab6;  
  
import java*.*awt*.*geom*.*Rectangle2D;  
  
public class BurningShip extends FractalGenerator {  
 public static final int LIMIT = 2000;  
  
 public void getInitialRange(Rectangle2D*.*Double range) {  
 range*.*x = -2;  
 range*.*y = -2.5;  
 range*.*width = 4;  
 range*.*height = 4;  
 }  
  
 public int numIterations(double x, double y) {  
 ComplexNum cmplx = new ComplexNum(0, 0);  
 int iterator = 0;  
  
 while (iterator < LIMIT && cmplx*.*getSquaredModule() < 4) {  
 cmplx*.*makeSquaredWithAbsInPoint(x, y);  
  
 iterator++;  
 }  
  
 if (iterator == LIMIT) return -1;  
  
 return iterator;  
 }  
  
 @Override  
 public String toString() { return "Burning Ship"; }  
}

package lab6;  
  
public class ComplexNum {  
 public double rl;  
 public double im;  
  
 public ComplexNum(double rl, double im){  
 this*.*rl = rl;  
 this*.*im = im;  
 }  
  
 public double getSquaredModule() {  
 return (this*.*rl \* this*.*rl + this*.*im \* this*.*im);  
 }  
  
 public void makeSquaredInPoint(double x, double y) {  
 double real = (rl \* rl) - (im \* im) + x;  
 double imagine = 2 \* rl \* im + y;  
  
 rl = real;  
 im = imagine;  
 }  
  
 public void makeSquaredWithConjInPoint(double x, double y) {  
 double real = (rl \* rl) - (im \* im) + x;  
 double imagine = - 2 \* rl \* im + y;  
  
 rl = real;  
 im = imagine;  
 }  
  
 public void makeSquaredWithAbsInPoint(double x, double y) {  
 double real = (rl \* rl) - (im \* im) + x;  
 double imagine = 2 \* Math*.*abs(rl) \* Math*.*abs(im) + y;  
  
 rl = real;  
 im = imagine;  
 }  
}

package lab6;  
  
import java*.*awt*.*\*;  
import javax*.*swing*.*\*;  
import javax*.*swing*.*filechooser*.*FileFilter;  
import javax*.*swing*.*filechooser*.*FileNameExtensionFilter;  
import java*.*awt*.*geom*.*Rectangle2D;  
import java*.*awt*.*event*.*\*;  
import java*.*awt*.*image*.*BufferedImage;  
import java*.*io*.*File;  
  
public class FractalExplorer {  
 private int displaySize;  
  
 private JImageDisplay display;  
  
 private FractalGenerator fractal;  
  
 private Rectangle2D*.*Double range;  
  
 private int rowRemaining;  
  
 private JButton saveButton;  
 private JButton resetButton;  
 private JComboBox comboBox;  
  
 public FractalExplorer(int size) {  
 displaySize = size;  
  
 fractal = new Mandelbrot();  
 range = new Rectangle2D*.*Double();  
  
 fractal*.*getInitialRange(range);  
 display = new JImageDisplay(displaySize, displaySize);  
 }  
  
 public void createAndShowGUI() {  
 display*.*setLayout(new BorderLayout());  
  
 resetButton = new JButton("Reset");  
 Resetter resetHandler = new Resetter();  
 resetButton*.*addActionListener(resetHandler);  
  
 saveButton = new JButton("Save");  
 Saver saveHandler = new Saver();  
 saveButton*.*addActionListener(saveHandler);  
  
 Clicker click = new Clicker();  
 display*.*addMouseListener(click);  
  
 FractalGenerator mandelbrotFractal = new Mandelbrot();  
 FractalGenerator tricornFractal = new Tricorn();  
 FractalGenerator burningShipFractal = new BurningShip();  
  
 comboBox = new JComboBox();  
  
 comboBox*.*addItem(mandelbrotFractal);  
 comboBox*.*addItem(tricornFractal);  
 comboBox*.*addItem(burningShipFractal);  
  
 Chooser fractalChooser = new Chooser();  
 comboBox*.*addActionListener(fractalChooser);  
  
 JLabel label = new JLabel("Fractal:");  
  
 JPanel panel = new JPanel();  
 panel*.*add(label);  
 panel*.*add(comboBox);  
  
 JPanel myBottomPanel = new JPanel();  
 myBottomPanel*.*add(saveButton);  
 myBottomPanel*.*add(resetButton);  
  
 JFrame myFrame = new JFrame("Fractal Explorer");  
  
 myFrame*.*setDefaultCloseOperation(JFrame*.*EXIT\_ON\_CLOSE);  
 myFrame*.*add(myBottomPanel, BorderLayout*.*SOUTH);  
 myFrame*.*add(display, BorderLayout*.*CENTER);  
 myFrame*.*add(panel, BorderLayout*.*NORTH);  
  
 myFrame*.*pack();  
 myFrame*.*setVisible(true);  
 myFrame*.*setResizable(false);  
 }  
  
 private void drawFractal() {  
 enableUI(false);  
  
 rowRemaining = displaySize;  
  
 for (int y = 0; y < displaySize; y++){  
 FractalWorker drawRow = new FractalWorker(y);  
 drawRow*.*execute();  
 }  
 }  
  
 private void enableUI(boolean value) {  
 comboBox*.*setEnabled(value);  
 resetButton*.*setEnabled(value);  
 saveButton*.*setEnabled(value);  
 }  
  
 private class Resetter implements ActionListener {  
 public void actionPerformed(ActionEvent e) {  
 if (e*.*getActionCommand()*.*equals("Reset")) {  
 fractal*.*getInitialRange(range);  
 drawFractal();  
 }  
 }  
 }  
  
 private class Chooser implements ActionListener {  
 public void actionPerformed(ActionEvent e) {  
 Object source = e*.*getSource();  
 if (source instanceof JComboBox) {  
 JComboBox comboBox = (JComboBox) source;  
  
 fractal = (FractalGenerator) comboBox*.*getSelectedItem();  
 assert fractal != null;  
  
 fractal*.*getInitialRange(range);  
 drawFractal();  
 }  
 }  
 }  
  
 private class Saver implements ActionListener {  
 public void actionPerformed(ActionEvent e) {  
 if (e*.*getActionCommand()*.*equals("Save")) {  
 JFileChooser fileChooser = new JFileChooser();  
  
 FileFilter extensionFilter = new FileNameExtensionFilter(  
 "PNG",  
 "png"  
 );  
  
 fileChooser*.*setFileFilter(extensionFilter);  
  
 fileChooser*.*setAcceptAllFileFilterUsed(false);  
  
 int userSelection = fileChooser*.*showSaveDialog(display);  
  
 if (userSelection == JFileChooser*.*APPROVE\_OPTION) {  
 java*.*io*.*File file = fileChooser*.*getSelectedFile();  
 String filePath = file*.*getPath();  
  
 if (!filePath*.*contains(".png")) file = new File(filePath + ".png");  
 try {  
 BufferedImage displayImage = display*.*getImage();  
 javax*.*imageio*.*ImageIO*.*write(displayImage, "png", file);  
 } catch (Exception exception) {  
 JOptionPane*.*showMessageDialog(display,  
 exception*.*getMessage(), "Cannot Save Image",  
 JOptionPane*.*ERROR\_MESSAGE);  
 }  
 }  
 else return;  
 }  
 }  
 }  
  
 private class Clicker extends MouseAdapter {  
 @Override  
 public void mouseClicked(MouseEvent e) {  
 if (rowRemaining != 0) return;  
  
 int x = e*.*getX();  
 double xCoord = FractalGenerator*.*getCoord(range*.*x,  
 range*.*x + range*.*width, displaySize, x);  
  
 int y = e*.*getY();  
 double yCoord = FractalGenerator*.*getCoord(range*.*y,  
 range*.*y + range*.*height, displaySize, y);  
  
 fractal*.*recenterAndZoomRange(range, xCoord, yCoord, 0.5);  
  
 drawFractal();  
 }  
 }  
  
 private class FractalWorker extends SwingWorker<Object, Object> {  
 int row;  
 int[] rgbValues;  
  
 private FractalWorker(int row) { this*.*row = row; }  
  
 @Override  
 protected Object doInBackground() {  
 rgbValues = new int[displaySize];  
  
 for (int x = 0; x < rgbValues*.*length; x++) {  
 double xCoord = FractalGenerator*.*getCoord(range*.*x,  
 range*.*x + range*.*width, displaySize, x);  
  
 double yCoord = FractalGenerator*.*getCoord(range*.*y,  
 range*.*y + range*.*height, displaySize, row);  
  
 int iteration = fractal*.*numIterations(xCoord, yCoord);  
  
 if (iteration == -1) {  
 display*.*drawPixel(x, row, 0);  
 } else {  
 float hue = 0.5f + (float) iteration / 50;  
 int rgbColor = Color*.*HSBtoRGB(hue, 1f, 1f);  
  
 rgbValues[x] = rgbColor;  
 }  
 }  
 return null;  
 }  
  
 protected void done() {  
 for (int x = 0; x < rgbValues*.*length; x++) {  
 display*.*drawPixel(x, row, rgbValues[x]);  
 }  
  
 display*.*repaint(0, 0, row, displaySize, 1);  
  
 rowRemaining--;  
 if (rowRemaining == 0) enableUI(true);  
 }  
 }  
  
 public static void main(String[] args)  
 {  
 FractalExplorer displayExplorer = new FractalExplorer(800);  
 displayExplorer*.*createAndShowGUI();  
 displayExplorer*.*drawFractal();  
 }  
}

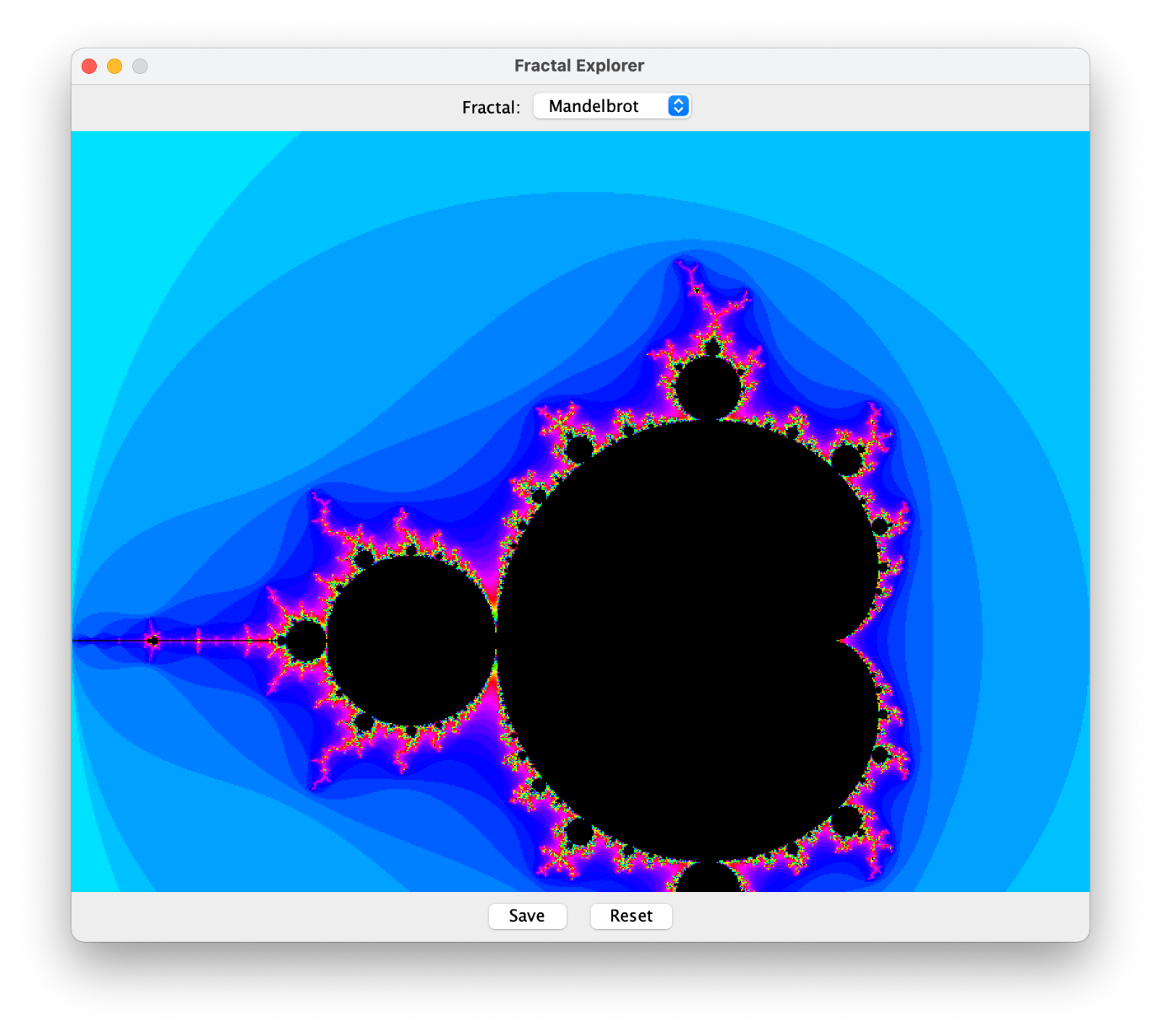
package lab6;  
  
import java*.*awt*.*geom*.*Rectangle2D;  
  
  
*/\*\*  
 \* This class provides the common interface and operations for fractal  
 \* generators that can be viewed in the Fractal Explorer.  
 \*/*public abstract class FractalGenerator {  
  
 */\*\*  
 \* This static helper function takes an integer coordinate and converts it  
 \* into a double-precision value corresponding to a specific range. It is  
 \* used to convert pixel coordinates into double-precision values for  
 \* computing fractals, etc.  
 \*  
 \** ***@param*** *rangeMin the minimum value of the floating-point range  
 \** ***@param*** *rangeMax the maximum value of the floating-point range  
 \*  
 \** ***@param*** *size the size of the dimension that the pixel coordinate is from.  
 \* For example, this might be the image width, or the image height.  
 \*  
 \** ***@param*** *coord the coordinate to compute the double-precision value for.  
 \* The coordinate should fall in the range [0, size].  
 \*/* public static double getCoord(double rangeMin, double rangeMax,  
 int size, int coord) {  
  
 assert size > 0;  
 assert coord >= 0 && coord < size;  
  
 double range = rangeMax - rangeMin;  
 return rangeMin + (range \* (double) coord / (double) size);  
 }  
  
  
 */\*\*  
 \* Sets the specified rectangle to contain the initial range suitable for  
 \* the fractal being generated.  
 \*/* public abstract void getInitialRange(Rectangle2D*.*Double range);  
  
  
 */\*\*  
 \* Updates the current range to be centered at the specified coordinates,  
 \* and to be zoomed in or out by the specified scaling factor.  
 \*/* public void recenterAndZoomRange(Rectangle2D*.*Double range,  
 double centerX, double centerY, double scale) {  
  
 double newWidth = range*.*width \* scale;  
 double newHeight = range*.*height \* scale;  
  
 range*.*x = centerX - newWidth / 2;  
 range*.*y = centerY - newHeight / 2;  
 range*.*width = newWidth;  
 range*.*height = newHeight;  
 }  
  
  
 */\*\*  
 \* Given a coordinate <em>x</em> + <em>iy</em> in the complex plane,  
 \* computes and returns the number of iterations before the fractal  
 \* function escapes the bounding area for that point. A point that  
 \* doesn't escape before the iteration limit is reached is indicated  
 \* with a result of -1.  
 \*/* public abstract int numIterations(double x, double y);  
}

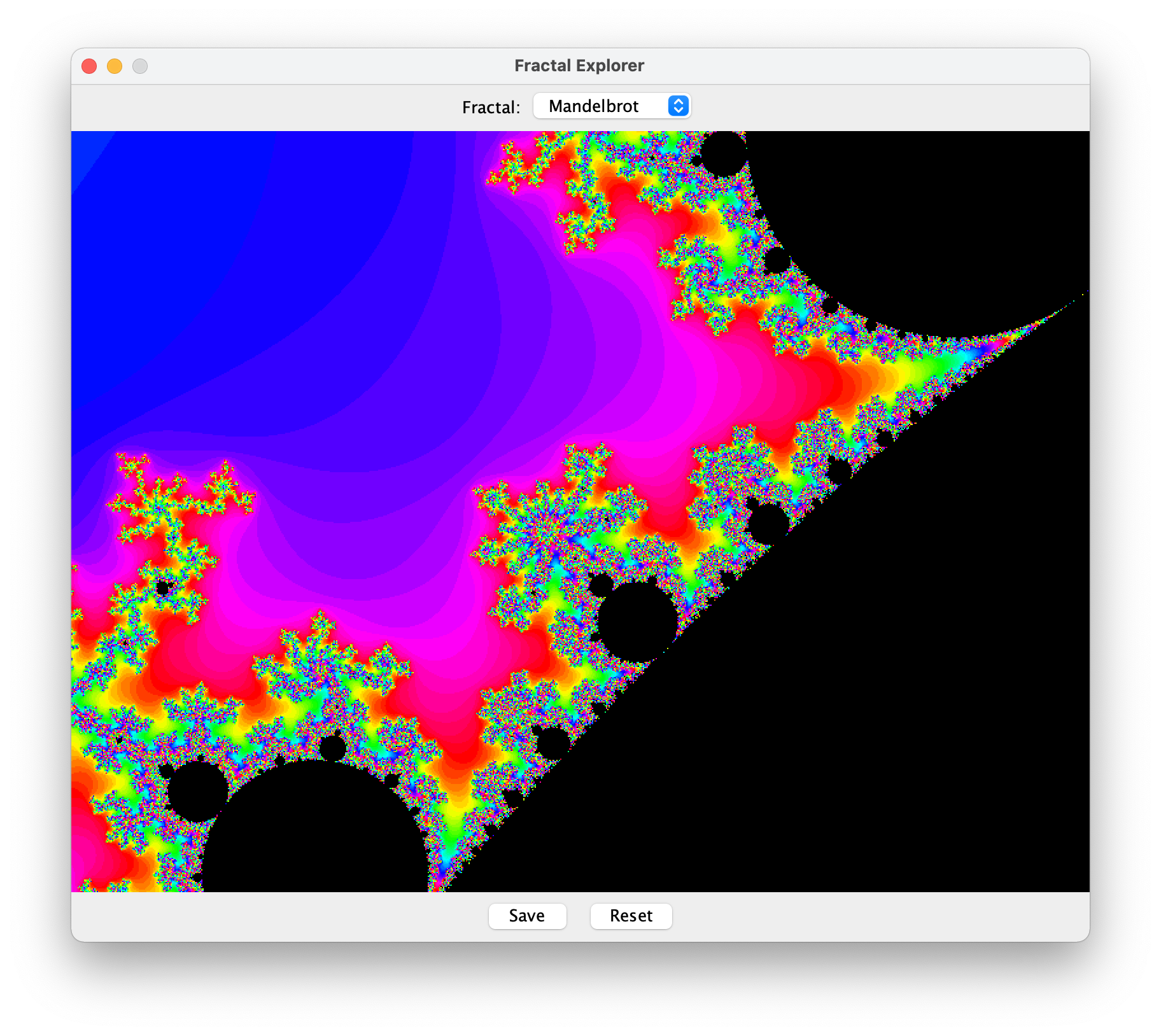
package lab6;  
  
import javax*.*swing*.*\*;  
  
public class FractalWorker extends SwingWorker {  
 @Override  
 protected Object doInBackground() throws Exception {  
 return null;  
 }  
}

package lab6;  
  
import javax*.*swing*.*JComponent;  
import java*.*awt*.*\*;  
import java*.*awt*.*image*.*BufferedImage;  
  
public class JImageDisplay extends JComponent {  
 private final BufferedImage image;  
  
 public BufferedImage getImage() {  
 return image;  
 }  
  
 public JImageDisplay(int w, int h){  
 if (w <= 0)  
 throw new IllegalArgumentException("w must be > 0; got " + w);  
  
 if (h <= 0)  
 throw new IllegalArgumentException("h must be > 0; got " + h);  
  
 image = new BufferedImage(w, h, BufferedImage*.*TYPE\_INT\_RGB);  
 Dimension dimension = new Dimension(w, h);  
  
 super*.*setPreferredSize(dimension);  
 }  
  
 @Override  
 protected void paintComponent(Graphics g) {  
 super*.*paintComponent(g);  
  
 g*.*drawImage (image, 0, 0, image*.*getWidth(), image*.*getHeight(), null);  
 }  
  
 public void clearImage() {  
 Graphics2D imageGraphics = image*.*createGraphics();  
 imageGraphics*.*setColor(new Color(0, 0, 0));  
  
 imageGraphics*.*fillRect(0, 0, image*.*getWidth(), image*.*getHeight());  
 }  
  
 public void drawPixel (int x, int y, int rgbColor){  
 image*.*setRGB(x, y, rgbColor);  
 }  
}

package lab6;  
  
import java*.*awt*.*geom*.*Rectangle2D;  
  
public class Mandelbrot extends FractalGenerator {  
 public static final int LIMIT = 2000;  
  
 public void getInitialRange(Rectangle2D*.*Double range) {  
 range*.*x = -2;  
 range*.*y = -1.5;  
 range*.*width = 3;  
 range*.*height = 3;  
 }  
  
 public int numIterations(double x, double y) {  
 ComplexNum cmplx = new ComplexNum(0, 0);  
 int iterator = 0;  
  
 while (iterator < LIMIT && cmplx*.*getSquaredModule() < 4) {  
 cmplx*.*makeSquaredInPoint(x, y);  
  
 iterator++;  
 }  
  
 if (iterator == LIMIT) return -1;  
  
 return iterator;  
 }  
  
 @Override  
 public String toString() { return "Mandelbrot"; }  
}

package lab6;  
  
import java*.*awt*.*geom*.*Rectangle2D;  
  
public class Tricorn extends FractalGenerator {  
 public static final int LIMIT = 2000;  
  
 public void getInitialRange(Rectangle2D*.*Double range) {  
 range*.*x = -2;  
 range*.*y = -2;  
 range*.*width = 4;  
 range*.*height = 4;  
 }  
  
 public int numIterations(double x, double y) {  
 ComplexNum cmplx = new ComplexNum(0, 0);  
 int iterator = 0;  
  
 while (iterator < LIMIT && cmplx*.*getSquaredModule() < 4) {  
 cmplx*.*makeSquaredWithConjInPoint(x, y);  
  
 iterator++;  
 }  
  
 if (iterator == LIMIT) return -1;  
  
 return iterator;  
 }  
  
 @Override  
 public String toString() { return "Tricorn"; }  
}

****

****